下载APP

08 | 什么办法可以在讨论中自然形成统一语言？

2021-07-10 徐昊

《如何落地业务建模》 课程介绍 >

讲述：徐昊

时长 22:18 大小 20.42M

你好，我是徐昊。今天我们来聊聊事件建模法（Event-based modeling）。

对于大多数人而言，业务建模中最难的一步并不是获得模型，而是说服业务方接受模型作为统一语言。虽然我们上节课讲到可以把角色 - 目标 - 实体法当作一种共创方法，但在实际操作的过程中，角色 - 目标 - 实体法仍然存在收集 - 建模 - 说服这三步。那么，有没有一种方法，可以在讨论的过程中，更自然地完成模型共创呢？

答案是肯定的。**事件建模法就是这样一种更易于模型共创的方法**。不同于原味面向对象范式关注实体之间的关联与交互，事件建模法通过事件捕捉系统中信息的改变，再发掘触发这些改变的源头，然后通过这些源头发现背后参与的实体与操作，最终完成对系统的建模。

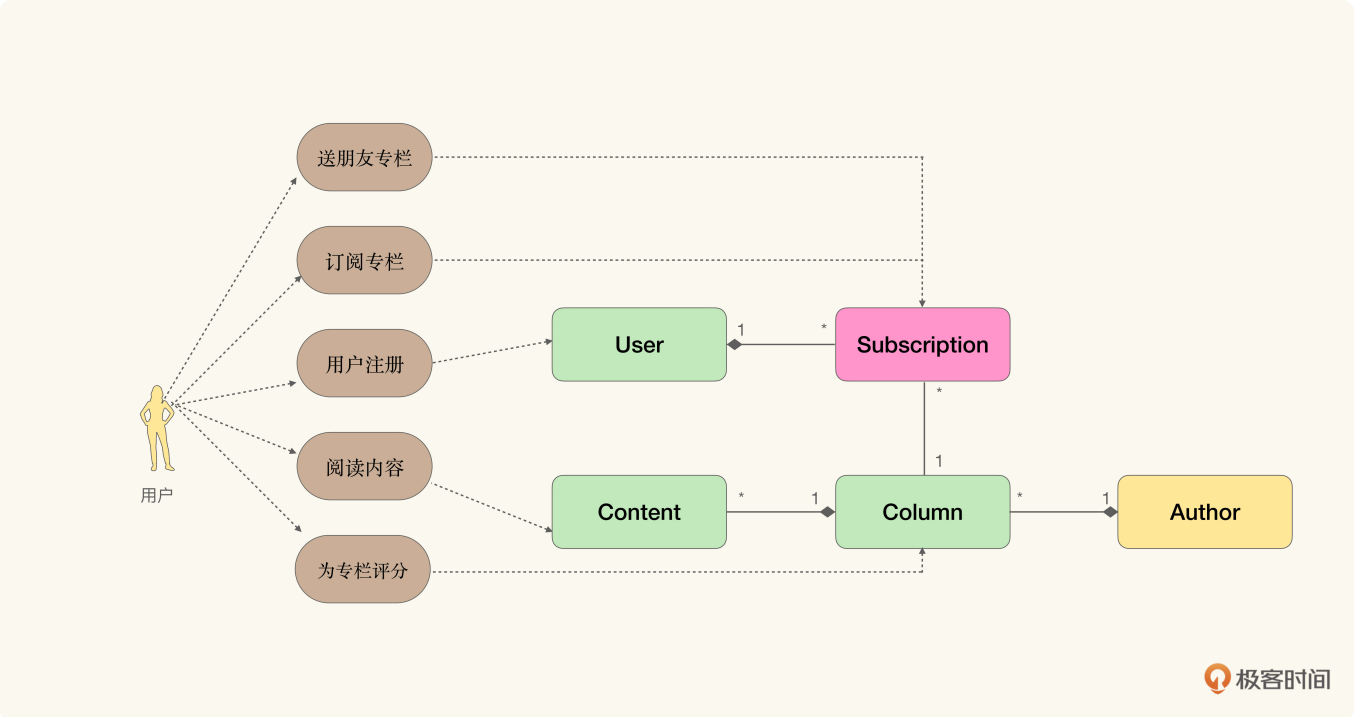
目前有两种比较有代表性的事件建模法，一种是目前 DDD 社区热捧的事件风暴法（Event Storming），另一种是我从 Peter Coad 的彩色建模中演化出的四色建模法。这节课我们先来学习事件风暴法，下节课我再展开讲解四色建模法。

不过在学习这两种具体的建模方法之前，我们有必要先了解事件建模法的 2 个基本原则，分别是通过事件表示交互，和通过时间线划分不同事件。

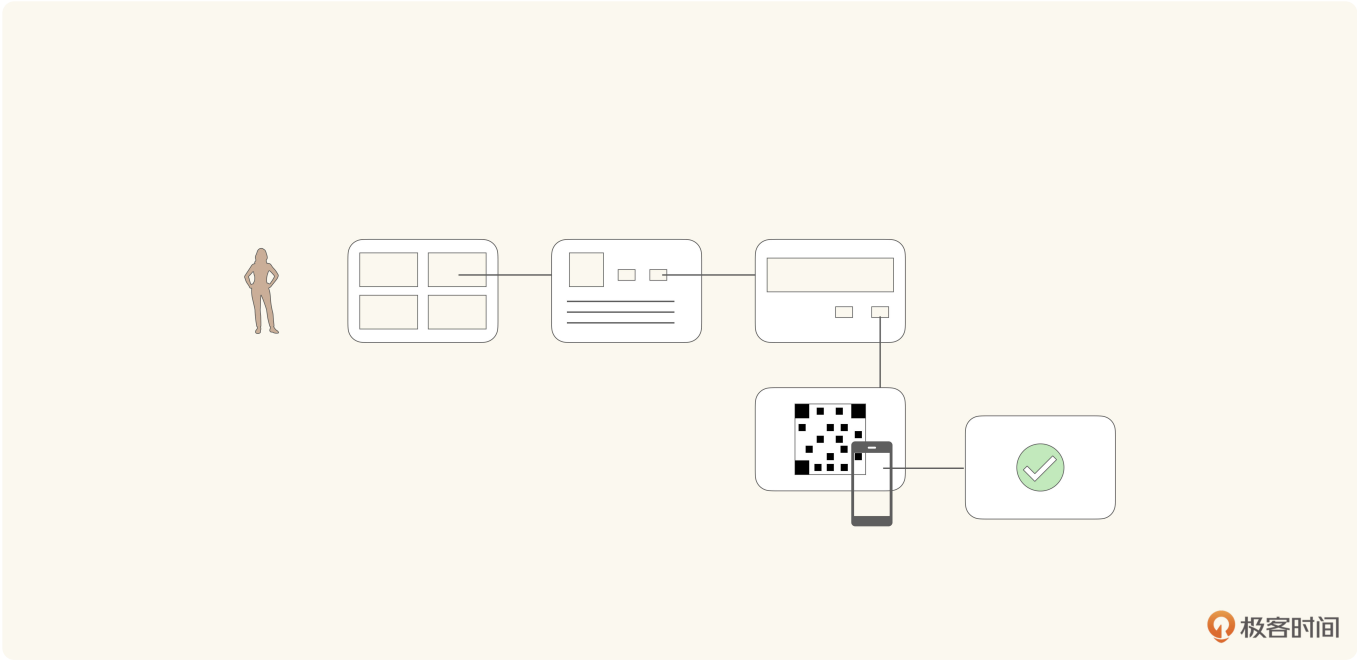
事件建模法的基本原则（1）：通过事件表示交互

在催化剂方法中，我们将交互直接以对象的形式引入到了领域模型中。这虽然在一定程度上展开了隐藏在领域模型中的业务维度，但是在多数情况下，结果的粒度仍嫌太粗。这也是为什么**催化剂方法只能用于简单项目，或是项目的初始阶段**。

而且随着项目的深入进行，它能提供的信息就变得相当有限了。我们回顾一下之前的催化剂模型，订阅专栏这个流程被表示为了一个交互对象。



或许对于简单的流程来说，催化剂方法已经足够了。但如果订阅专栏流程本身就包含很多步骤，那么展开的信息就不足以帮助业务方作出有益的决定了。如下图所示的流程，就是从专栏列表中挑选心仪专栏，查看详情后通过手机购买的过程。



那你肯定会有疑问了。如果只有一个交互对象，业务方仍然不知道在具体的步骤中，接触到了哪些领域模型，以及在具体的步骤中，领域模型所包含的信息是否能够支撑业务的实现。那么我们**要怎样展开业务流程，才能让它以恰当的粒度自然地融入领域模型呢？**

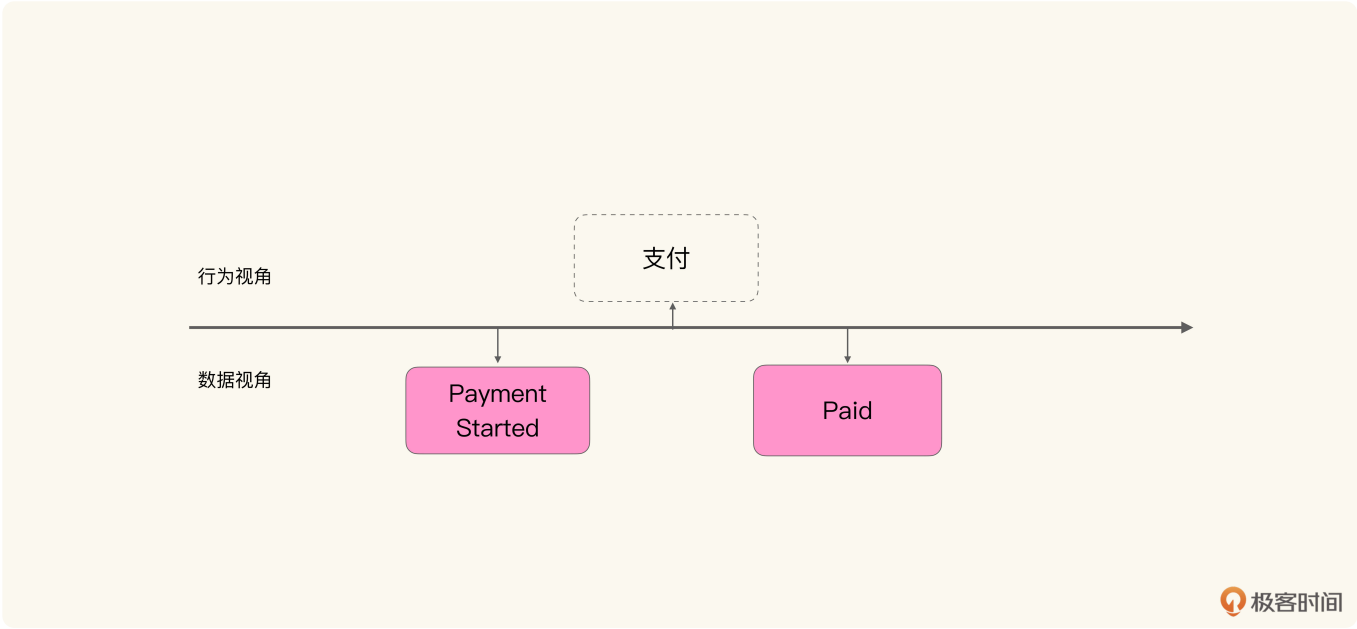
这里的难点有两个：**第一个是恰当的粒度；第二个是融入领域模型**。毕竟我们上节课已经讲过了，对于催化剂方法的一大诟病就是交互的对象难以直接与软件实现关联。

我们先来解决第二个问题，因为第一个问题的解法与它有关。第二个问题的核心困难在于该如何通过对象模型表达流程。如前面所讲，**模型偏重于数据角度**，描述了在不同业务维度下，数据将会如何改变，以及如何支撑对应的计算与统计。而业务维度中的**流程、交互、功能点，则更关注行为**。

那我们**该怎么弥合数据与行为之间的差异呢**？事件（Event）是一种行之有效的办法。

我们可以**把事件看作行为的印记**。比如支付这个行为。我们不需要直接描述支付这个行为，而是通过捕捉这个行为前后的事件：支付发起（Payment Started）和支付完成（Paid）。要知道，事件自身能表达的含义有限，但是将一系列事件按照发生顺序排列起来，就能还原发生过的行为。

因而**事件发生的时间点**就是事件最重要的属性。我们会在模型中画一条时间轴，并依据事件发生的时间点，按先后顺序将事件标记在时间轴上。这样的话，一前一后成对出现的支付发起与支付完成，就暗示着在两个事件之间，存在一个成功的支付行为：



接下来就可以根据事件去寻找背后的领域模型了。根据我自己的经验，我们可以**按照记叙文六要素去想：时间、地点、人物、起因、经过和结果。**

经过是我们需要表示的行为，不需要出现在模型里；

时间可以是事件顺序的依据；

地点、人物、起因、结果，则是寻找领域概念的依据。

比如上面支付这个例子：

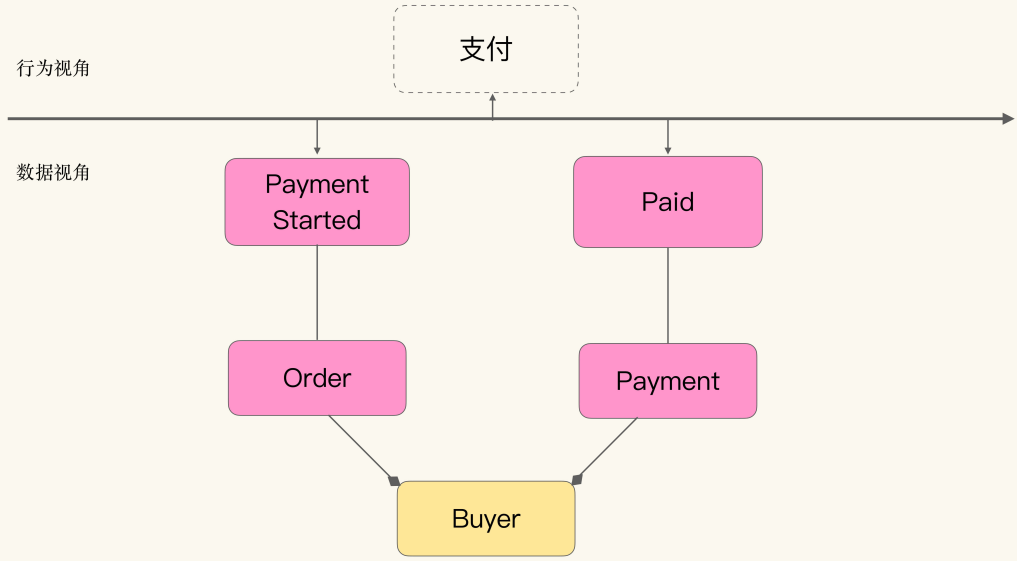
地点在这里不太需要；

人物就是买家（Buyer）；

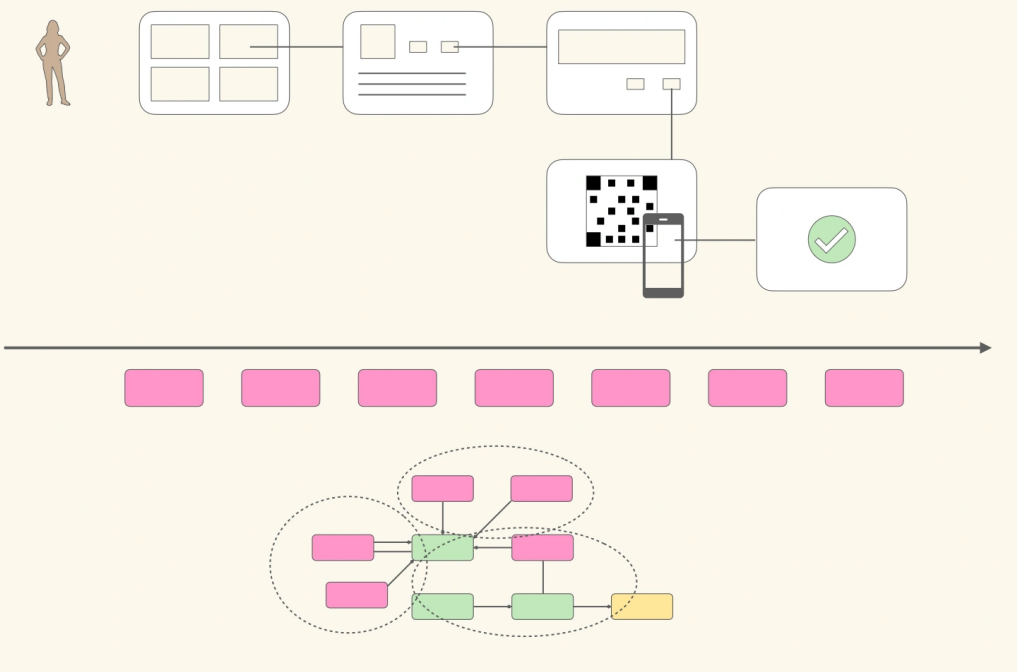
起因是订单（Order），因为需要要支付的是订单，那么我们可以把起因事件（Payment Started）和订单关联；

结果是支付凭证（Payment），我们将它与结果事件（Paid）关联。

当我们再补全模型之间的关联，就能得到这样一个更完整的模型了：



那么我们只需要根据交互流程或者是业务流程，寻找出对应的事件，并以事件为指引，就能找到事件背后的领域模型了。下面是一个示意图，表明这个过程和结果。



通过这个例子，我们可以看到事件建模法与其他面向对象建模法的差别，这些差异带来了三个优点。也正是这三个优点，使得**事件建模法更容易与业务方完成模型共创**。

第一，虽然看起来多出了一个额外的寻找事件环节，但对于大多数项目而言，**建立业务流程或交互流程都是必需的**。如果是从头开始的绿地项目（Greenfield），就需要建立流程以明确需求；如果是存量项目，从流程开始梳理功能也是必备的步骤。

那么在梳理流程的同时发现事件，或者是直接通过事件描述流程，都是惠而不费的操作。顺便说一句，目前更常见的做法是直接通过事件描述流程。这种做法也是一种预热，让业务方可以提前适应从数据角度来描述流程。

第二，**通过记叙文六要素去寻找领域概念**，很自然地就关联了业务维度，提高了业务方对统一语言的接受度。当我们按照“时间、地点、人物、起因、经过、结果”还原“领域情节”，从中去寻找业务概念的时候，这些业务概念很自然地就连带了业务的维度。

毕竟我们的大脑善于记忆情节，如果某个概念出现在有情节的场景中，那么我们的大脑不光会记住这个概念，还会记住与这个概念相关的情节。当我们回想或是再听到这个概念时，就会下意识地回忆起相关的情节。

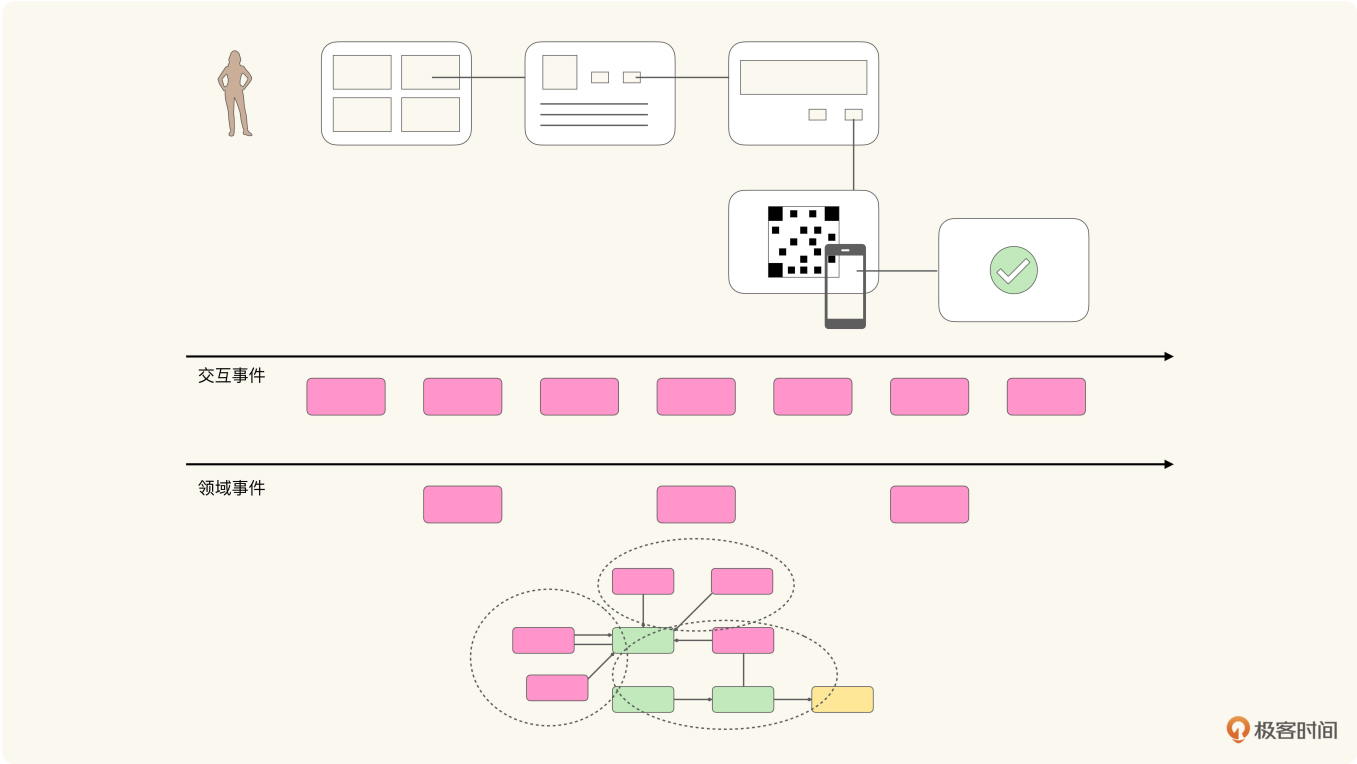
那么对于通过领域情节获取的领域概念，业务方不光会记住领域概念，也会记住领域情节。这种连带记忆，让我们很自然地把业务维度展开到了模型里。

第三，**事件建模对于最终得到的领域模型并没有什么额外的限制**，不会因为我们使用了事件建模法，就得到奇怪的模型。它只是一种更自然地发现领域概念的方式。而领域概念与领域逻辑的组织，还是交由建模者自行掌控。

事件建模法的基本原则（２）：通过时间线划分不同事件

通过事件的记叙性，我们已经将流程性的业务维度在模型中展开了。但是如果事无巨细地将所有交互都转化为事件，那么对模型似乎会有较大的干扰。这也是我们之前提到的问题，到底**怎么才能找到合适的粒度将业务维度展开到领域模型中**？

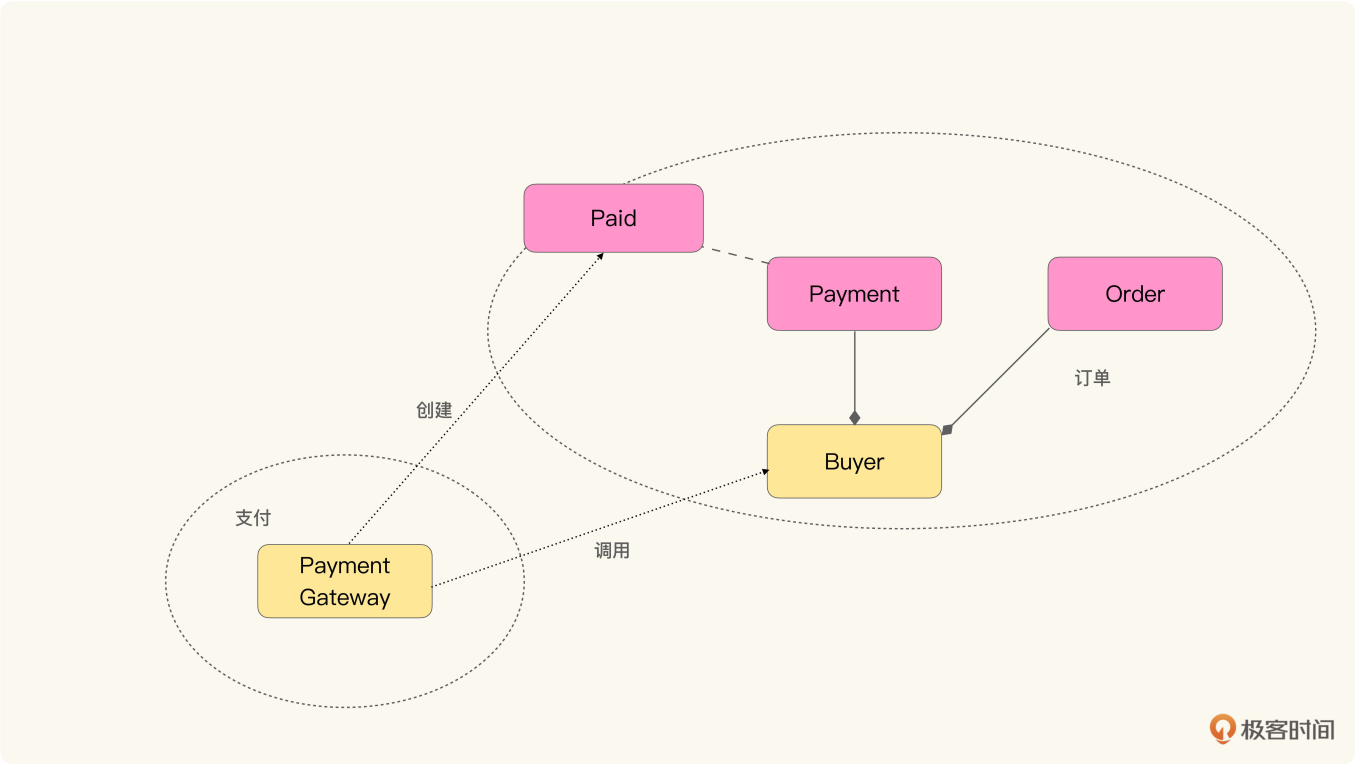
事件建模法有一个巧妙的办法解决了这个问题：**引入多条时间线**。一种常见的做法是，引入一条名叫领域事件的时间线，其上发生的都是对领域模型有重要意义和影响的事件。那么之前的时间线也就相应地被称作交互事件时间线：



交互事件我们很容易理解，也就是通过交互流程产生的事件，那么什么是领域事件呢？**领域事件**其实指的就是**发生在领域中且值得注意的事件。而领域事件通常意味着领域对象状态的改变**，因此这也是我们判断一个事件是否是领域事件的重要依据。能引起领域对象变更的就是领域事件，否则就是交互事件。

比如前面例子的两个事件：支付发起和支付完成。支付发起不会引起领域对象的变更，而支付完成则意味着会增加一个支付凭证。那么如果在这两个事件中去区分的话，我们更有理由选择支付完成作为一个领域事件，而将支付发起看作交互事件。

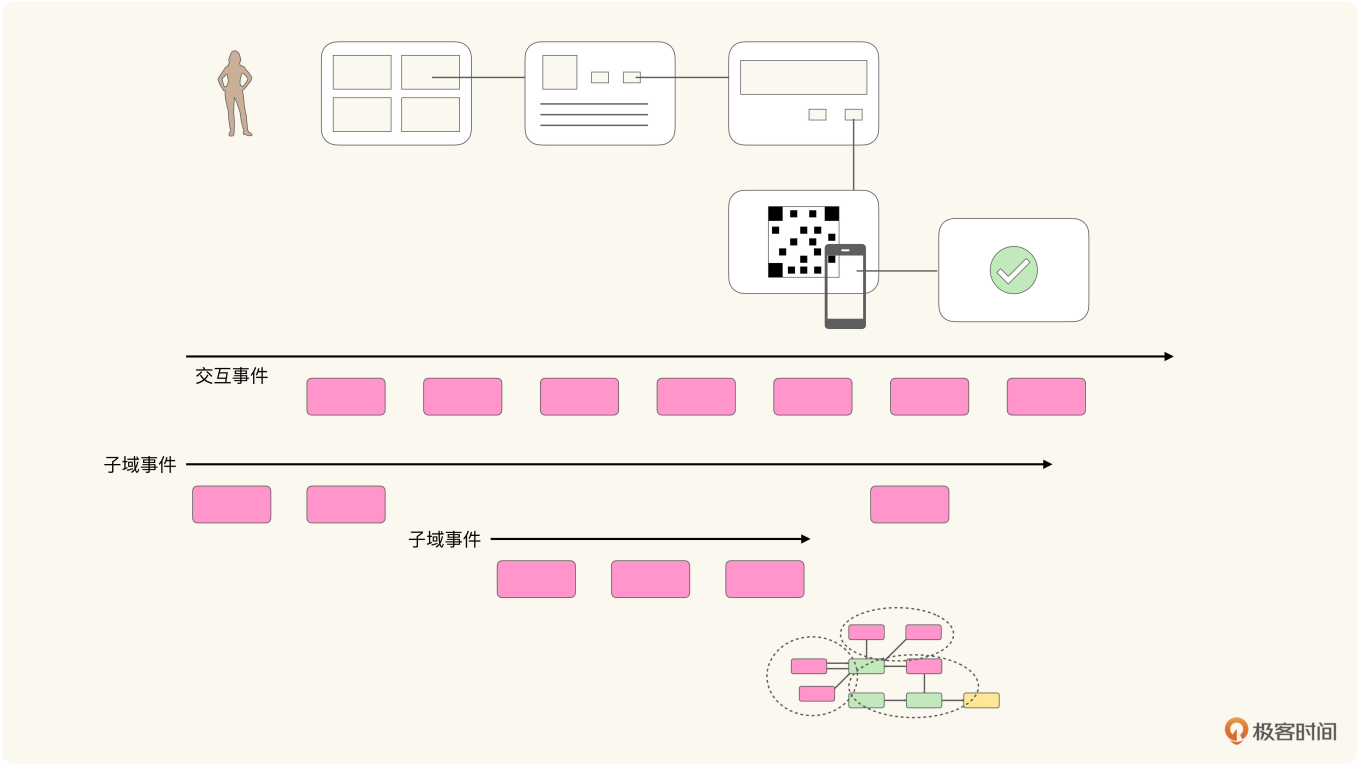
还有一个办法也可以用来判断某个事件是否是领域事件，就是**能否将这个事件看作某个领域逻辑的等价接口**。如果我们把订单上下文和支付上下文单拿出来看：



当支付完成的时候，支付网关希望通知订单上下文构建支付凭证。因为支付凭证（Payment）被角色买家（Buyer）聚合，那么支付网关需要调用 Buyer 的逻辑构建凭证。而如果把订单上下文当作一个整体，还可以发送事件支付完成（Paid）到订单上下文，通知订单上下文构建支付凭证。这时发到上下文上的事件就和聚合根的行为接口等价了。因而**可以起到这种等价作用的事件，就是领域事件。**

按照这两个办法过滤出的领域事件，在逻辑上就是一个恰当的粒度，可以帮助我们将业务中的流程维度在领域模型上展开。

除了能够划分交互事件与领域事件之外，我们还可以通过引入时间线帮助我们细化事件与上下文的关系，这是一种很有效的建模技巧。特别是当涉及跨多子域协同的时候，按子域分离时间线可以帮助我们更好地理解子域间的交互：



凡可归类为事件建模的建模方法在底层逻辑上都是类似的，都是通过寻找事件，以及事件背后的领域概念，完成对领域概念的挖掘和建模。

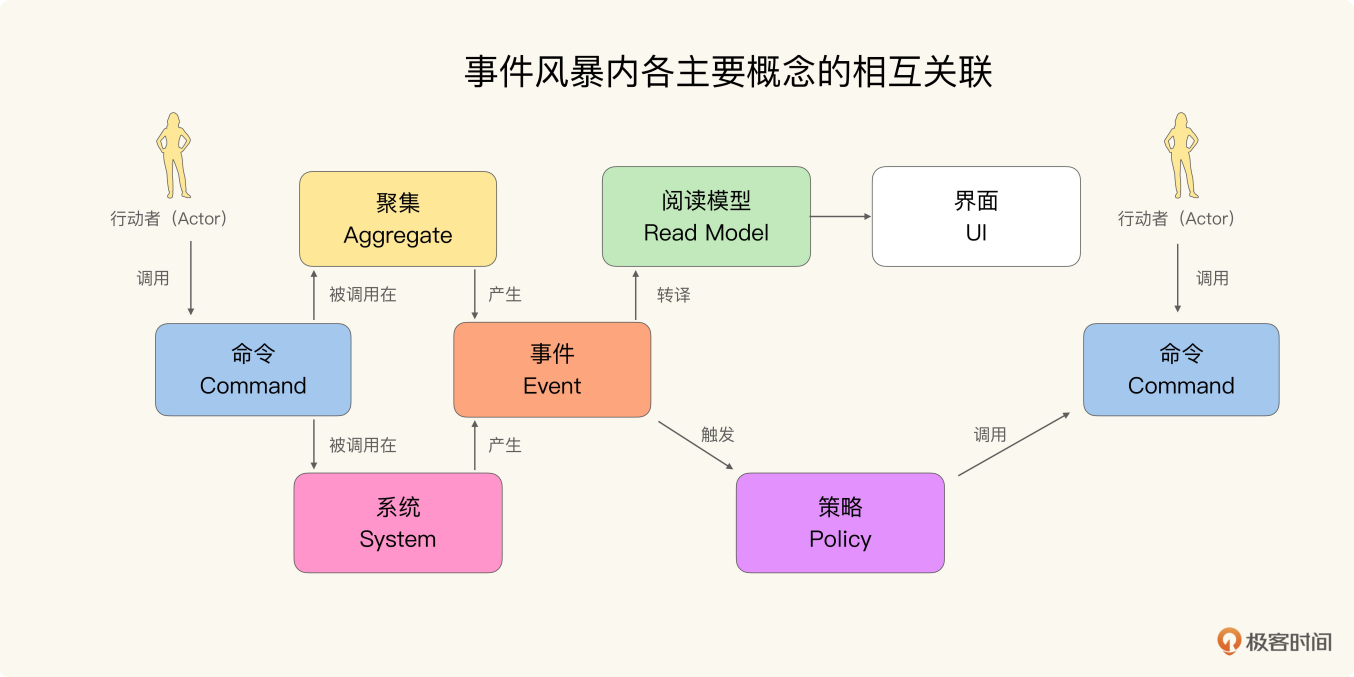
不同方法之间的差异就在于两点：**以何种逻辑发现事件；事件如何与模型结合**。抓住了这两点差异，我们就能很好地理解不同的事件建模法了。事件建模法是一种元方法（Meta Methodology），当你熟练使用之后，可以根据业务需要发明自己的方法。当然在那之前，你还需要参考学习别人是如何使用事件建模的，然后来构造自己的建模法。

接下来，我们就看一下事件风暴法（Event Storming）。它是目前最为流行的事件建模法了。请你着重体会我刚才说的这两点差异，看看它是以何种逻辑发现事件的，事件又是如何与模型结合的。

事件风暴法

事件风暴是意大利人 Alberto Brandolini 在 2012 年创造的一种事件建模方法。在形式上，事件风暴是一种互动式建模工作坊，通过将不同背景的项目参与方汇聚一堂，集思广益，从而形成有效的模型。这也是事件风暴名称的由来——因为它本质上就是一种头脑风暴（Brainstorming），按照欧美人士习惯的俏皮式命名法，将头脑（Brain）改成其建模法的关键元素事件（Event）从而得名。

从建模方法上来看，事件风暴是一种事件建模法。它以响应式编程（Reactive Programming）作为范式，通过事件、命令与策略之间的响应关系，组织逻辑。事件风暴内各主要概念的相互关联如下图所示：



乍一看这张图里有很多概念，大部分应该可以望文生义，但为了明晰概念，我还是解释一下：

行动者（Actors）是系统的使用者。这里使用者是一个相对模糊的概念，可能是现实中的人也可能是别的系统；

命令（Command）是由行动者发起的行为。它代表了某种决定，通常是事件的起因，也称作行动者触发命令（AIC，Actor Initiated Command）；

事件（Event）就是我们前文讨论过的事件；

聚集（Aggregate）就是领域驱动设计中的聚合，可以看作一组领域对象，在头脑风暴阶段可以泛指某些领域概念，不需要细化；

系统（System）指代的是不需要了解细节的三方系统。因为不需要了解细节，所以我们可以将它们看作一个整体；

阅读模型（Read Model）用以支撑决策的信息。通常与界面布局有关；

策略（Policy）是对于事件的响应，通常表示不属于某些聚集的逻辑。通过策略可以触发新的命令，由策略触发的命令，被称作系统触发命令（SIC，System Initiated Command）。

事件风暴内的主要概念

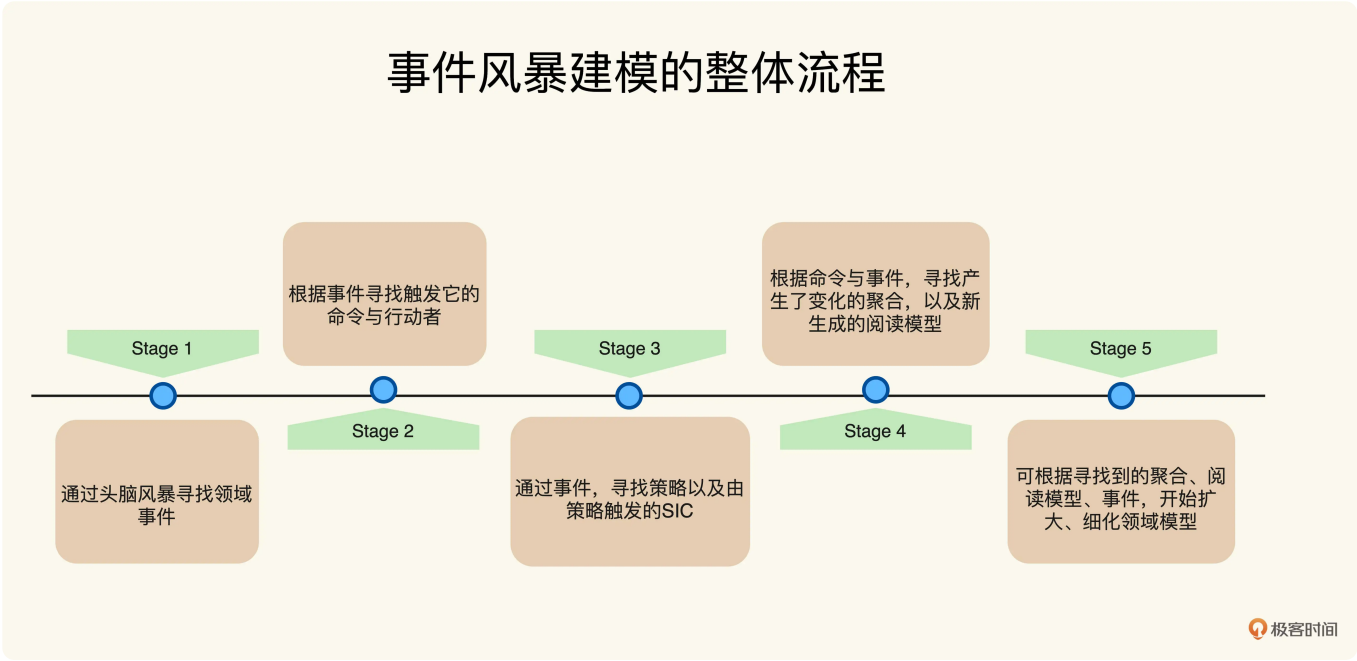
概念	含义
行动者（Actors）	是系统的使用者。这里使用者是一个相对模糊的概念，可能是现实中的人也可能是别的系统
命令（Command）	是由行动者发起的行为。它代表了某种决定，通常是事件的起因，也称作行动者触发命令（AIC，Actor Initiated Command）
事件（Event）	我们前文讨论过的事件
聚集（Aggregate）	就是领域驱动设计中的聚合，可以将其看作一组领域对象。在头脑风暴阶段，它可以泛指某些领域概念，不需要细化
系统（System）	指代的是不需要了解细节的第三方系统。因为不需要了解细节，所以我们可以将它们看作一个整体
阅读模型（Read Model）	用以支撑决策的信息，通常与界面布局有关
策略（Policy）	是对于事件的响应，通常表示不属于某些聚集的逻辑。通过策略可以触发新的命令，而由策略触发的命令，被称作系统触发命令（SIC，System Initiated Command）

极客时间

对于图中如此多的颜色我也要多说一句，事件风暴法是一种头脑风暴的形式，头脑风暴自然离不开 3M 便条纸（Post-It）。而我们尽可能多地通过色彩的维度来表示不同概念，是头脑风暴的一个窍门。因此，事件风暴法推荐使用 7 种不同的颜色，来表示这 7 个不同的概念。

事件风暴建模的整体流程是这样的：

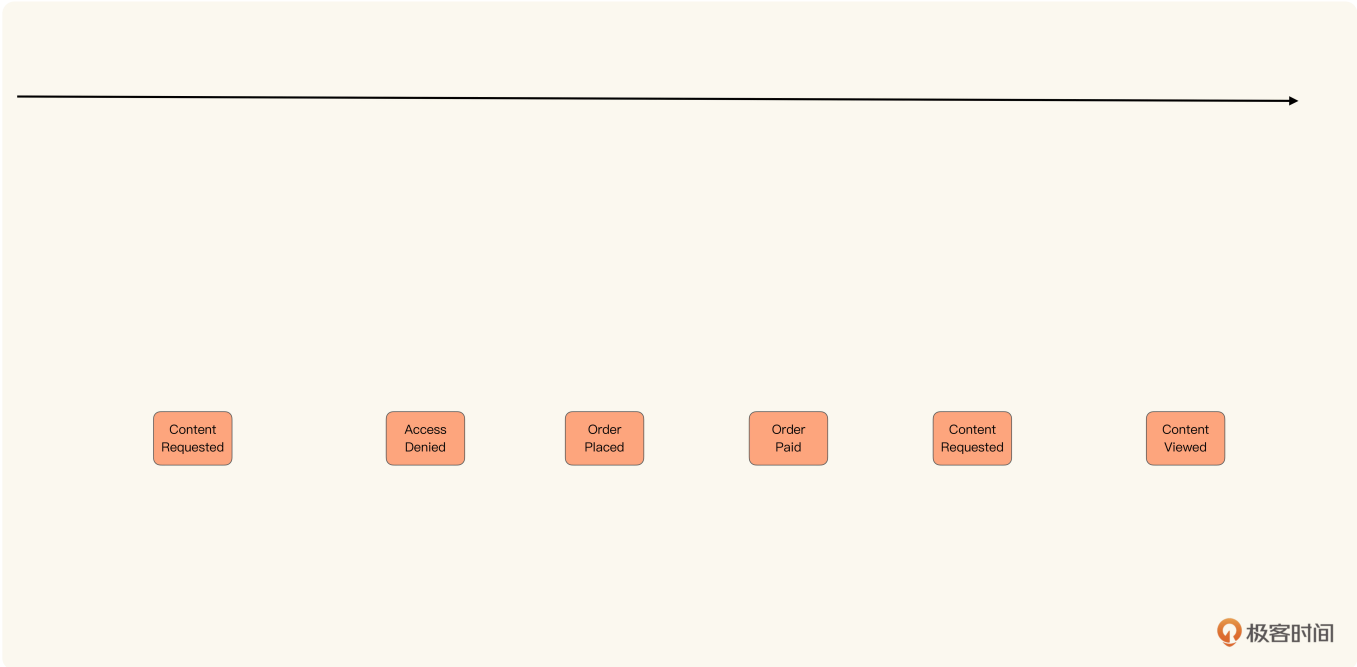
1. 首先通过头脑风暴寻找领域事件；
2. 根据事件寻找触发它的命令与行动者；
3. 通过事件，寻找策略以及由策略触发的 SIC；
4. 根据命令与事件，寻找产生了变化的聚合，以及新生成的阅读模型；
5. 根据寻找到的聚合、阅读模型、事件，开始完善、细化领域模型。



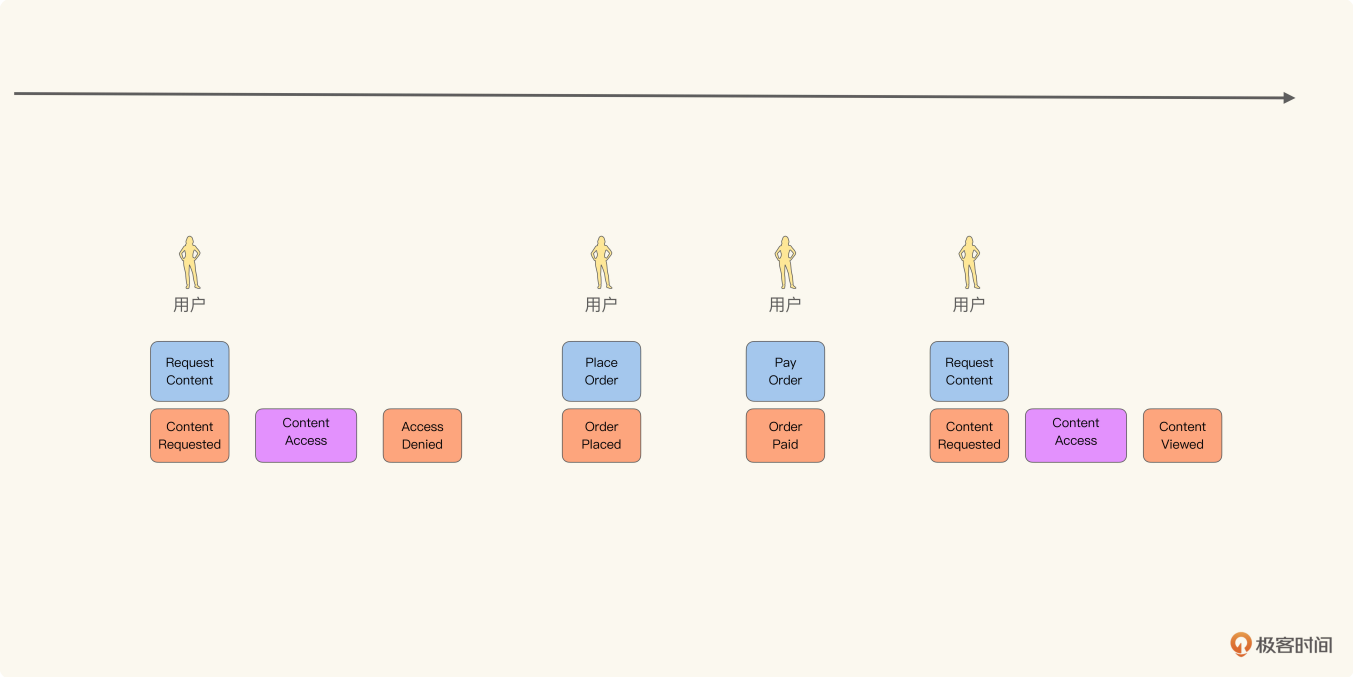
可以看到，事件风暴法的建模流程遵循事件建模的大体框架，它的特点就在于通过头脑风暴发现事件，再依赖触发与响应寻找事件间的关系，通过聚合与阅读模型寻找领域模型。下面让我们仍然以极客时间专栏订阅为例，展示一下事件风暴建模法的大致过程。

我们的场景是这样的：用户发现了想看的内容，但是因为没有订阅专栏看不了，于是下单订阅专栏，完成支付之后，再次访问之前内容，就能看到了。

首先我们从事件入手，根据上面的流程，很容易就能通过头脑风暴得到关键的领域事件：内容请求（Content Requested）、访问拒绝（Access Denied）、订单确认（Order Placed）、订单支付（Order Paid）、内容被访问（Content Viewed）。



这里我们不难发现一个规律，事件都是以“**名词 + 动词被动式**”表示的。在寻找到事件之后，我们需要判断哪些是 AIC，哪些是 SIC。AIC 由命令触发，而 SIC 则由策略触发。在上面这些事件里，访问拒绝和内容被访问是 SIC，其他都是 AIC。我们可以根据这个，寻找到它们各自对应的触发源：

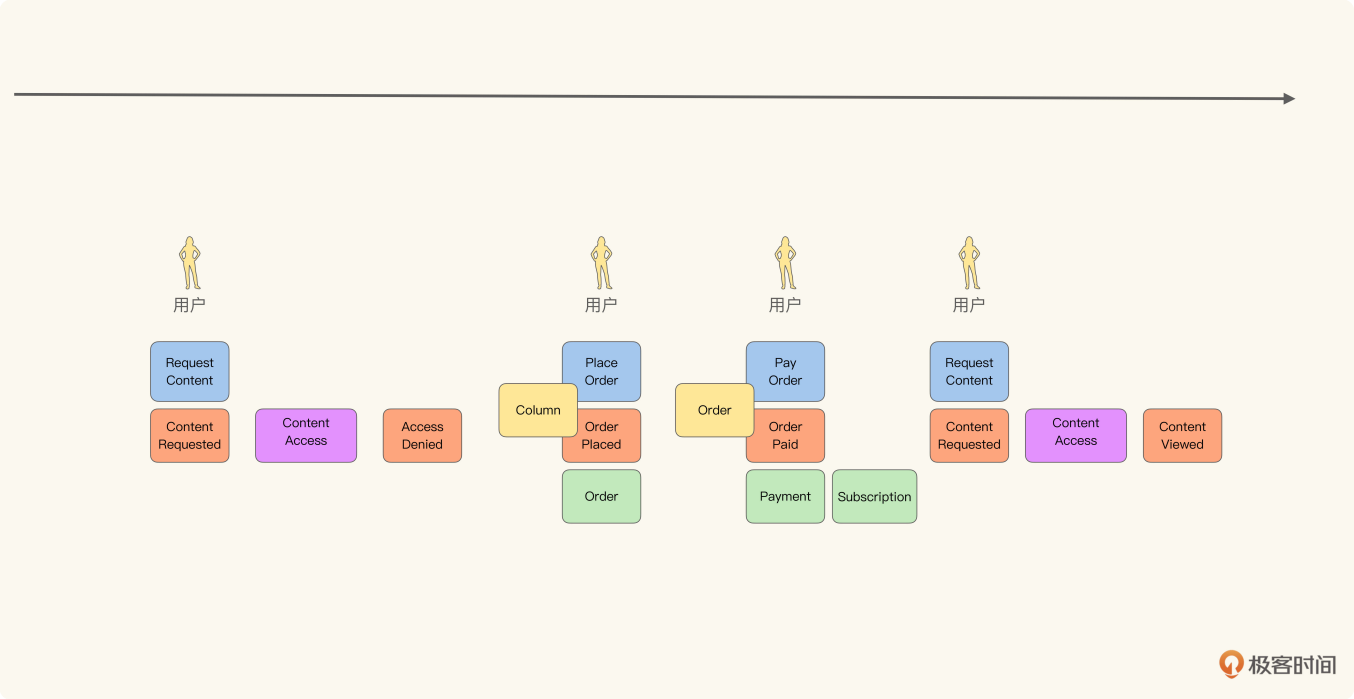


你会发现，命令的命名与事件正好相反，是“**动词 + 名词**”以表示对应事件的源头。策略的命名则按对应的业务逻辑命名即可。在这里我们称之为内容访问策略（Content Access Policy）。

确定了事件的源头，我们就可以开始寻找事件发生过程中，访问了哪些聚集，事件完成后生成了哪些阅读模型。

聚集在当前阶段并不需要细化为具体对象，可以泛指与某些功能有关的领域概念集合。比如在 Pay Order-Order Paid 这一组命令与事件中，我们可以知道它一定访问并修改了与订单（Order）相关的领域概念，那我们可以使用聚集 Order 泛指这些概念，而无须细化。

这个过程并不一定需要按照顺序，哪些事件对应的聚集比较明显，就可以先从它们入手。比如在上图中，与内容访问策略相关的阅读模型并不是非常明显，那不如先不考虑它，而是从其他更容易得出结论的部分入手：

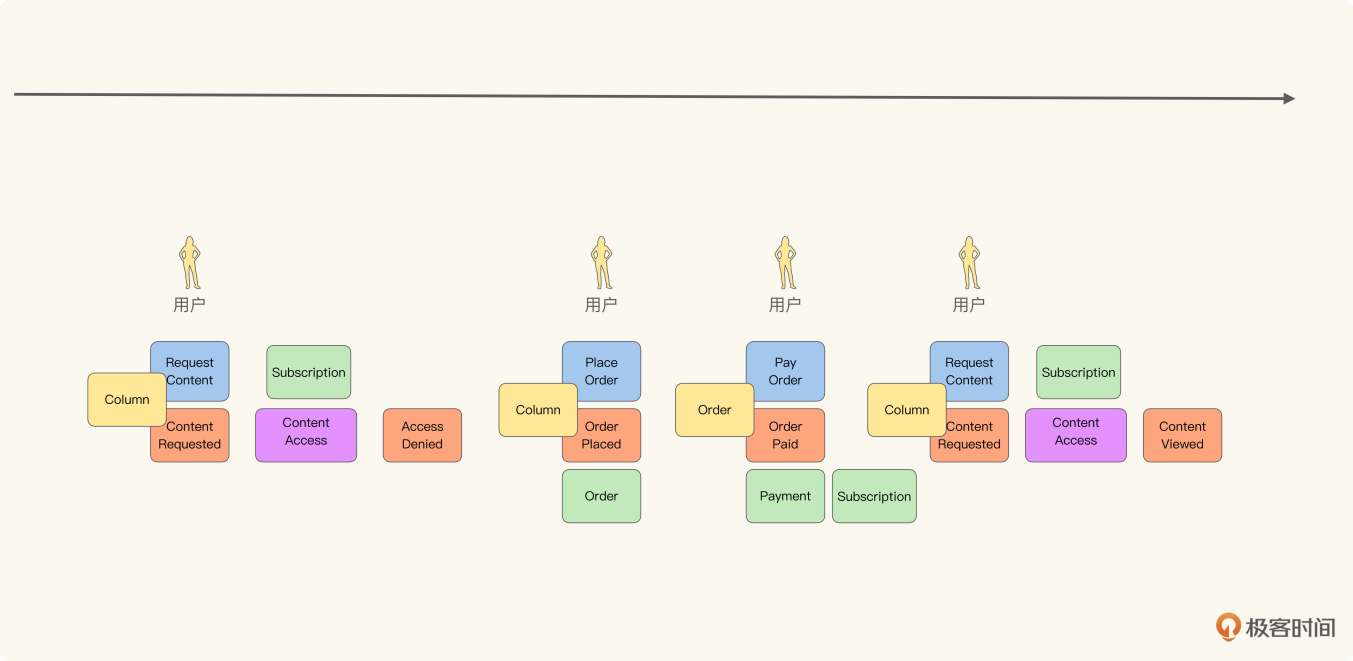


Place Order-Order Placed 产生的阅读模型是订单（Order），Pay Order-Order Paid 产生的阅读模型是订阅（Subscription）和支付（Payment）。这里我们会发现阅读模型与聚合有重合的地方。订单在 Pay Order-Order Paid 中成了聚合，而在 Place Order-Order Placed 里成了阅读模型。

阅读模型是比聚合更为宽泛的概念，可以看作聚合的超集。它既可以是聚合，也可以是为特定界面或报表生成的特定数据信息。这种做法是随着 NoSQL（Not Only SQL）兴起的一种反思——命令与查询职责分离（CQRS，Command Query Responsibility Separation）。

那么写入数据（Command）与查询数据（Query）一定要用同一个模型吗？原味面向对象的回答：是，应该一样。而越来越多的实践者则发现，分开可能更好。毕竟两者对于逻辑一致性的诉求不同，分开处理能得到更好的结果。因此在事件风暴建模法中，**阅读模型是包含写入与查询的所有数据形态的总集，而聚合只是阅读模型中符合对象风格的子集。**

当我们找到一些阅读模型后，再回过头去看之前留白不易处理的部分，就豁然开朗了。内容访问策略需要使用订阅阅读模型，决定了哪些内容对哪些用户可见：



在获取了聚合和阅读模型之后，我们就可以开始细化领域模型了。这时候方法与原味对象方法就没有什么不同了，我就不再赘述了。

通过这个简单的例子，相信你对事件风暴法的基本流程已经有所了解了：它通过头脑风暴发现领域事件，以“对于事件的响应”为主要维度寻找事件间的关联；它通过阅读模型和聚集发现事件与领域模型之间的关联。不难发现，事件风暴法是一种简单明快的事件建模方法。

小结

这节课我们先讲述了事件建模法作为一类方法的共同性质，也就是通过事件表示行为，再根据事件顺序去组织业务逻辑。并总结了这么做的好处：通过记述性描述发掘需求，不知不觉间构造领域模型，并完成统一语言。

之后我们以事件风暴法为例子，看到了这些原则在具体方法中是如何应用的。那么下一节中，我们来讲解另外一种事件建模法——四色建模法，在那之前我希望你思考以下问题。

思考题

事件的表现形式只能是事件吗？既有事件又存在阅读模型不是一种冗余吗？除了“事件 - 响应”外，还有什么办法可以展开维度？

通过“记叙文六要素”去寻找领域概念，很自然地就关联了业务维度，提高了业务方对语言的接受度。因为当我们按照“时间、地点、人物、起因、经过、结果”还原“领域情节”，寻找业务概念时，这些业务概念很自然地就连带了业务的维度。

如何落地业务建模

徐昊 | ThoughtWorks 中国区 CTO

识别二维码
免费试读

极客时间

欢迎把你的思考和想法分享在留言区，我会和你交流讨论。同时，我也会把其中不错的回答进行置顶。我们下节课再见！

分享给需要的人，Ta订阅后你可得 20 元现金奖励

赞 2 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 07 | 统一语言可以是领域模型本身吗？

下一篇 09 | 怎么才能更有效地获得事件流？

精选留言 (9)

写留言

冯 置顶

2021-07-14

我又想了一下，对于各种类型的系统，只要能找出可追溯的东西，比如钱的流向。再总结出一个合适的分析套路，就能形成一个这种类型的系统的分析方法

展开

作者回复: 嗯。你学会了

Oops!

2021-07-10

“Place Order-Order Placed 产生的阅读模型是订单（Order），Pay Order-Order Paid 产生的阅读模型是订阅（Subscription）和支付（Payment）。” 这个推导过程跨度有点大，这个阅读模型具体是怎么推导出来的呢？

作者回复: 根据产生了什么信息 被怎么读取 推理

2 3

hxfirefox
2021-07-16

对领域事件有疑问，在例子中content requested、content viewed看着都是个交互事件，而非领域事件，这样在事件风暴寻找领域事件应该如何去理解？

展开

作者回复: 与领域模型有关的就是领域事件 无关是交互。所以建模之前都是交互事件

1

邓志国
2021-07-18

事件事件记录的就是各种应该被记录的单据，有点像业务的操作日志。对对象状态变化，一种是修改对象，产生日志；一种是创建日志，进而推出状态变化。事件我觉得是记录了操作日志。

作者回复: 这种东西。法律要求必须记

梦倚栏杆
2021-07-18

对于策略类（内核架构）的系统，怎么来分析新需求呢？有没有规范化的套路，老师有什么好的建议吗？

作者回复: 找到一个隐喻 把问题简化

Jxin
2021-07-14

本章理解
1.事件意味着一种因果关系，这就使得一个静态的概念，却具备流动的张力。在识别和理解事件时，可以考虑为什么要产生这一事件，以及有什么要响应这一事件，进而思考推出产生事件的前序动作以及响应事件的后续动作，从而驱动参与者的“心流”不断思考下去，不断展开沟通。...
展开

作者回复: 应用事件也会产生数据。但不会涉及领域模型数据

冯
2021-07-14

事件风暴是一种团队协作方式，通过还原系统的运行轨迹发现关键的领域概念，而事件的发生顺序就是对系统行为的追溯。所以是不是凡是可追溯的东西，都能作为展开的维度？

事件和阅读模型的关系，事件是发生了一个动作的标志，这个动作还会影响系统，进而产生数据。阅读模型的一个功能就是这种数据的外在表现
展开

作者回复: nice

狩月
2021-07-12

订单上下文是以buyer聚合的， 必须要这样聚合吗？ 如果直接访问订单呢？ 这个聚合根的选择有什么方法没

作者回复: 请看前面的课 有讲到

赵晏龙
2021-07-11

所以【阅读模型】在系统实现中我是不是可以理解成【ViewModel视图模型】？
还有就是，这些建模方法确实都在从不同的方面去探查并对模型进行画像。那么，这些方

法的优劣和适用场景呢？是否可以加一些您的想法和理解？如果担心表述过于片面主观，可以加上【我认为】或者强调【我在实践中】这样的字眼。

展开

作者回复: 不行 view model不会出现在领域模型里

1